Dynaamisia metatietoja hallinnoiva sovellus Vaadin-kehyksellä

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu Riihimäki, Tieto- ja viestintätekniikan insinööri

Sykysy, 2017

Santtu Heleä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikan insinööri
Riihimäki

Tekijä Santtu Heleä **Vuosi** 2017

Työn nimi Dynaamisia metatietoja hallinnoiva sovellus Vaadin-kehyksellä

Työn ohjaaja/t Marko Hursti, Jari Mustajärvi

TIIVISTELMÄ

Opinnäytetyön toimeksiantajana toimi suomalainen ohjelmistotalo Triplan Oy, joka on erikoistunut julkishallinnon asian- ja dokumenttienhallinnan järjestelmiin. Heidän päätuotteensa, Tweb, joka on selainkäyttöinen asian- ja asiakirjahallinnan työväline, tarvitsisi sovellusta, jolla voitaisiin hallita Twebin dynaamisia metatietoja. Opinnäytetyön tavoitteena oli selvittää pilotoimalla, että olisiko kyseistä sovellusta mahdollista toteuttaa.

Opinnäytetyössä käyn läpi projektin tavoitteet ja tehtävät, sekä käsittelen Vaadin sovelluskehyksen kannalta keskeisiä asioita ja hyödyntämiäni tekniikoita. Jaan sovelluksen läpikäynnin kahteen eri kerrokseen, käyttöliittymäkerrokseen, sekä liiketoiminnalliseen kerrokseen. Käyttöliittymäkerroksessa painotetaan enemmän visuaalisia ratkaisuja, sekä yleistä käytettävyyttä käyttöliittymään liittyen, kun taas liiketoiminnallisessakerroksessa käydään läpi sovelluksen toimintaperiaatteet ja logiikka. Lopuksi teen yhteenvedon mahdollisista jatkokehitysehdotuksista, sekä niiden toteuttamisesta. Lisäksi, tarkastelen, että päästiinkö opinnäytetyössä sille asetettuihin tavoitteisiin.

Lopputuloksena saatiin aikaan pilotoitu sovellus, joka täytti sille asetetut vaatimukset ja tehtävät. Sovelluksesta käy selvästi ilmi, että Twebin dynaamisille metatiedoille pystyttäisiin kehittämään niitä hallinnoiva sovellus. Sovellus tehostaisi ja edistäisi työprosessia. Pilotoitu sovellus on kuitenkin vielä kaukana varsinaisesta käyttöönotosta, sillä sovellusta pitää testata varsinaisessa käytössä. Sovellus sai myös muutamia jatkokehitysehdotuksia, jotka tulee toteuttaa, ennen varsinaista käyttöönottoa.

Avainsanat Java, Vaadin, sovelluskehitys

Sivut 26 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology
Riihimäki

Author Santtu Heleä **Year** 2017

Subject Implementing an application to manage dynamic metadata using Vaadin Framework

Supervisors Marko Hursti, Jari Mustajärvi

ABSTRACT

Triplan Ltd was the commissioner of this thesis. Triplan Ltd is a Finnish software company, which specializes in the case and document management systems of public administration. Their main product, Tweb, which is a browser-based case and document management tool, required an application that could manage the dynamic metadata of Tweb. The goal of the project was to conduct a project and assess whether it was possible to implement such an application.

In this thesis, I go through the goals and tasks of the project. I also cover the essential terms and techniques used in the Vaadin Framework. I have divided the examination of the application into two different layers: the user interface layer and the business logic layer. In the user interface layer, I go through the thought process undergone and the decisions taken to improve user experience and to implement a clear and easy to use UI. As in the business logic layer, I explain the basic usage concepts of the application and the logic of its functions. Lastly, I wrap up the development suggestions and describe how they could be implemented. As an addition, I examine if the set goals were reached in the project.

As an end result, a pilot application was acquired which met the set requirements. Application shows that it is totally possible to develop an application to manage the dynamic metadata of Tweb. The application makes the work process more efficient and helps in the progress of managing dynamic metadata of Tweb. The application itself is still far away from being used as a daily tool. It needs to be tested in a real working environment by an actual user, not by the developer of the application. Also, the development recommendations should be evaluated and implemented before considering actual use of the application as a daily tool.

Keywords Java, Vaadin, development of applications

Pages 26 pages

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aihe-ehdotus tuli toimeksiantajalta, Triplan Oy:ltä, jossa suoritin myös työharjoittelua. Heillä oli tarve sovellukselle, jolla pystyttäisiin hallinnoimaan dynaamisia metatietoja, joita Triplan Oy:n päätuote Tweb hyödyntäisi. Tweb on selainkäyttöinen järjestelmä, joka on tarkoitettu asioiden ja dokumenttien hallintaan. Triplan Oy on ohjelmistotalo, joka keskittyy tarjoamaan tehokkaita ratkaisuja tiedonhallintaan ja tiedonhakuun. Triplanin yleisimmät asiakkaat ovat julkishallinnon organisaatioita, esimerkiksi kaupungit, sairaanhoitopiirit ja valtiohallinnon virastot, sekä laitokset.

Metatieto on varsinaista tietoa kuvaileva tieto. Julkishallinnon metatietomäärityksiä on saatavilla mm. JHS-suosituksista (julkisen hallinnon suositukset) ja SÄHKE2-normissa (Kansallisarkiston sähköisen säilyttämisen normi).
Tweb-järjestelmässä metatiedot ovat SÄHKE2-normin mukaiset, sillä järjestelmä on SÄHKE2-sertifioitu. Esimerkkejä metatiedoista ovat: asian nimike, asiakirjan laatimisaika, version laatija jne. Kaikki aineisto Tweb-järjestelmässä on luokiteltua tietoa (tiedonohjausjärjestelmän ohjaamaa) ja luokitus kuvaa tiedon tyyppiä. Tyyppejä voivat olla esimerkiksi: sopimus, hankinta-asia, viranhaltijapäätös tai rakennuslupa. Tweb-järjestelmän muuttuva, dynaaminen metatieto on kytketty näihin tyyppeihin. Sopimustyypillä on siis käytettävissä sopimusten dynaamiset, muuttuvat metatiedot. Tämä dynaamisuus ei ole suositusten tai normien vaatimus, vaan tulee asiakasvaatimuksista.

Opinnäytetyö on toiminnallinen projekti, jonka lopputuloksena on selainpohjainen sovellus. Toiminnallisuus siis koostuu sovelluksen suunnittelusta ja tuottamisesta ennalta määritetyin toiminnollisuuksiin. Tästä syystä tutkimusmenetelmät olivat hyvin vähäiset, ja kyseinen osuus koostuukin Vaadin sovelluskehyksen ja eri teknologien, kuten Javan ja SQL:n opiskelusta. Tämä antoi tiedon ja taidon soveltaa ja hyödyntää kyseisiä oppeja sovelluksen suunnittelussa ja toteuttamisessa.

2 PROJEKTIN TAVOITTEET JA TEHTÄVÄT

Projektin tavoitteena on pilotoida selainpohjainen sovellus Triplan Oy:n päätuotteen, Twebin, dynaamisten metaluokkien hallintaan. Sovelluksesta voisi graafisen käyttöliittymän kautta valita tiettyjä metaluokkia tai sen alaluokkia, ja luoda niistä automaattisesti lisäämistä varten vaadittavat skrip-

tit erillisenä tiedostona. Sovellus tulisi sisäiseen käyttöön ja tehostaisi työprosessia, kun eri asiakkaille luodaan metaluokkia. Sovellus toteutettaisiin hyödyntäen Vaadin sovelluskehystä.

Ennen kaikkea, sovellus mahdollistaisi kaikkien asiakkaiden metaluokkien ja niiden alaluokkien löytymisen yhdestä konkreettisesta paikasta. Tällä hetkellä metaluokkia luodaan excel-taulukkopohjilla, joista saadaan ulos SQL-skriptit, jotka tietokantaan ajettaessa luovat kyseiset metaluokat. Taulukoita on työlästä ylläpitää, ja kun asiakkaita on useita, niin on myös taulukoitakin. Sovelluksen käyttöliittymän tulisi olla helppokäyttöinen ja selkeä.

Sovelluksen suunnittelun edetessä, sekä muutamien kokouksien jälkeen, sovelluksen toiminnallisuus rajattiin tiettyihin mittoihin. Tämä on ensimmäinen yritys ja lähestymistapa ratkomaan kyseistä ongelmaa, joten kyseessä on enemmänkin prototyyppi, kuin lopulliseksi sellaisenaan päätyvä sovellus. Myös sovelluskehys, jota käytettäisiin olisi Vaadin Framework 8. Aikaisemmat Vaadin sovelluskehystä hyödyntävät sovellukset yrityksen alla käyttävät Vaadin Framework 7: aa.

Perustoiminnoiltaan sovelluksen pitäisi pystyä tuottamaan samanlaisia skriptejä, kuin aikaisemmin on excel-taulupohjilla luotu. Näin vältettäisiin ongelmatilanteet Twebin yhteensopivuuden kanssa. Koska tarkoituksena olisi tuottaa samanlaista jälkeä kuin excel-taulupohjillakin. Näin varmistetaan myös sovelluksen skriptien toimivuus ajettaessa kantaan, sillä excel-taulupohjilla luodut skriptit toimivat moitteettomasti, niin sovelluksessakin luodut toimisivat. Sovelluksen pitäisi myös näyttää kaikki metaluokat ja niiden alaluokat eli attribuutit. Tämä mahdollistaisi excel-taulukoista pois siirtymisen, ja kaikki tarvittava tieto löytyisi sovelluksen tietokannasta, mihin kaikki pääsisivät helposti käsiksi.

Käyttöliittymän tulisi myös olla helppokäyttöinen ja selkeä. Metaluokkien ja attribuuttien lisääminen, sekä poistaminen tulisi olla selkeää ja helppoa käyttäjälle. Sovellus ehdottaisi erinäisten tietojen arvoja automaattisesti, jotta käyttäjän ei tarvitsisi vaivata päätä kyseisten tietojen kanssa. Tällaisia tietoja olisivat muun muassa yksilöivä id-arvo, sekä järjestysnumero.

3 VAADIN

Vaadin Oy on suomalainen ohjelmistoalan yritys, jonka päätuote on Vaadin Framework -kehitysympäristö. Yritys perustettiin vuonna 2000, jolloin yrityksen nimenä oli IT Mill Oy. Yritys vaihtoi nimekseen kuitenkin Vaadin Oy vuonna 2011. Nimenmuutosta perustelee Vaadin Oy:n toimitusjohtaja Joonas Lehtinen seuraavasti: ”Meille ratkaiseva tekijä nimenmuutoksessa oli, kun huomasimme, että erityisesti Suomen ulkopuolella oli vaikeaa

saada ihmisiä liittämään IT Mill ja Vaadin toisiinsa. Tuote oli ihmisten mielissä selkeästi tärkeämpi asia kuin sen takana oleva yritys”(Vaadin 2011).

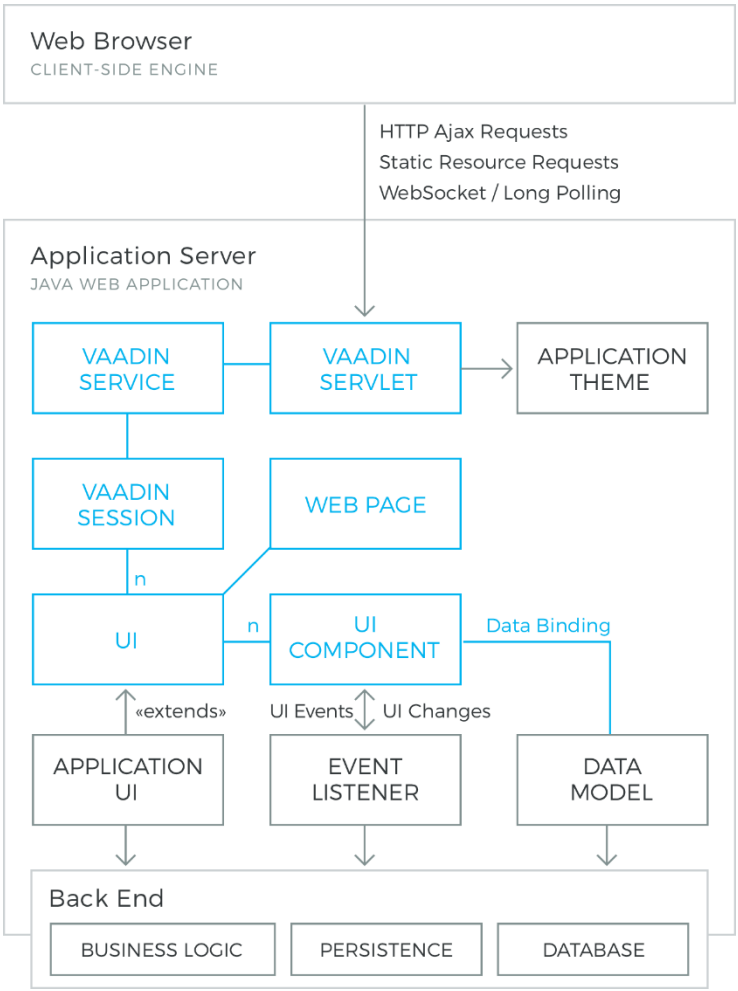
Yrityksellä on Suomessa toimistot Turussa ja Helsingissä, sekä ulkomailla Saksassa ja Yhdysvalloissa. Yritys työllistää yli 140 työntekijää ja heidän liikevaihtonsa on 11 miljoonan dollarin paremmalla puolella. (Vaadin 2017a.)

Yrityksen menestys on heidän Vaadin Framework -kehitysympäristön ansiota. Vaadin Framework perustuu avoimeen lähdekoodiin, jonka ansiosta tuote on levinnyt ympäri maailmaa. Avoimen lähdekoodin ansiosta Vaadin Frameworkin ympärillä toimii laaja aktiivinen yhteisö. Yhteisö käyttää Vaadin-teknologiaa ja kehittää Vaadinta päivä päivältä julkaisten lisäkomponentteja tuotteelle, jotka yritys voi sitten halutessaan liittää osaksi varsinaista Vaadin Frameworkia. (Vaadin 2017a.)

3.1 **Vaadin Framework**

Vaadin Framework eli Vaadin sovelluskehys on avoimen lähdekoodin, Java-ohjelmointikieltä käyttävä, sovelluskehys, jolla voidaan luoda rikkaita internet-sovelluksia. Rikkaalla internet-sovelluksella (Rich Internet Application, RIA) tarkoitetaan internet-sovellusta, joka toiminnaltaan vastaa hyvinkin pitkälti normaalia työpöytäsovellusta, mutta sitä ajetaan asiakkaan internet-selaimen sisällä. Tämän ansiosta RIA:t eivät yleensä vaadi ohjelmiston asentamista asiakaspuolelle laisinkaan. Rikkaat internet-sovellukset voidaan ajaa, joko täysin asiakkaan selaimessa, niin kuin GWT:n (Google Web Toolkit) tapauksissa, tai palvelimella, jolloin käyttäjällä on käyttöliittymä selaimessaan ja selain keskustelee palvelimen kanssa. Vaadin sovelluksissa, vaihtoehto on yleensä jälkimmäinen. (Vaadin 2017b.) (Wikipedia 2017a.)

Rikkaassa internet-sovelluksessa prosessointi tapahtuu suurimmaksi osaksi selaimessa. Sovelluksen client-osat toimivat asiakkaan järjestelmässä eristetyssä ympäristössä, hiekkalaatikossa (sandbox). Tämä mahdollistaa asiakkaan (clientin) järjestelmän hyödyntämisen esimerkiksi sovelluksen lokaaleissa toiminnoissa ja laskennassa. Samalla myös tietoliikenne asiakas-palvelin suunnassa vähenee. Hiekkalaatikko-ympäristössä ajaminen myös edistää tietoturvallisuutta, sillä ympäristö rajoittaa tiedostoihin ja käyttöjärjestelmään pääsyä, sekä niiden näkyvyyttä palvelinpuolelle. (Rouse n.d.)



Kuva 1. Vaadin Framework Application Architecture (Vaadin, n.d.).

Kuva 1. esittää Vaadin sovelluskehyksellä tuotettujen web-sovellusten perusarkkitehtuurin. Arkkitehtuuri voidaan jakaa kahteen eri osaan: Palvelinpuolen kehykseen ja asiakaspuolen ”moottoriin”. Asiakaspuolen ”moottori” operoi selaimessa JavaScript-koodina. Pitää kuitenkin muistaa, että sovelluskehyksen käyttäjän ei tarvitse osata JavaScriptiä, vaan kirjoittaminen tapahtuu Javalla, minkä jälkeen sovelluskehys kääntää Javan JavaScriptiksi. Moottori renderöi selaimeen käyttöliittymän ja välittää käyttäjän toiminnot käyttöliittymässä palvelimelle. Palvelimella sovellus ajetaan Java Servlettinä. JavaScriptin ajaminen selaimessa tarkoittaa taas sitä, että asiakas ei tarvitse mitään lisäosia selaimeensa käyttääkseen Vaadin sovellusta. Tämä on selkeä hyöty Vaadin sovelluskehykselle, sillä muut kehykset, jotka perustuvat esimerkiksi Flashiin tai Java Applet:eihin vaativat selainlisäosia toimiakseen. (Vaadin Team 2017, 1.)

3.2 Ohjelmointimallit

Vaadin sovelluskehys on suunniteltu rikkaiden internet-sovellusten ja käyttöliittymien tuottamiseen. Toki sovelluskehyksellä voidaan tehdä myös pe-

rinteisempiä internetsivuja. Yritys kuitenkin asettaa Vaadin sovelluskehyksensä enemmänkin sovellusten kehittämiseen kuin nettisivujen kehittämiseen. Sovelluskehys tukee kahta eri ohjelmointimallia: palvelinpuoli (server-side) ja asiakaspuoli (client-side). (Vaadin Team 2017, 1.)

Näistä kahdesta mallista palvelinpuolen malli on tehokkaampi, sillä se mahdollistaa ohjelmoijan keskittyä täysin sovelluslogiikkaan. Palvelinpuolen sovelluskehys huolehtii AJAX kommunikoinnista selaimen ja palvelimen välillä, sekä huolehtii käyttöliittymän oikeasta käyttäytymisestä käyttäjän selaimessa. Tällöin ohjelmoijan ei tarvitse opetella ja välittää selainteknologioista, kuten HTML:stä tai JavaScript:stä. Vaadin hyödyntää Google Web Toolkittiä (GWT) selainyhteensopivuuden saavuttamiseksi, jolloin ohjelmoijan ei tarvitse huolehtia selaintuestakaan. Vaatimen tehokkuus perustuukin hyvin pitkälti siihen, että ohjelmoijalle mahdollistetaan keskittyminen täysin sovelluksen logiikkaan. HTML, JavaScript ja muu vastaava selainpainotteinen teknologia, kuten selaintuki ovat käytännössä näkymättömiä sovelluksen logiikalle. (Vaadin Team 2017, 2-3.)

Palvelinpuolen mallin lisäksi Vaadin tarjoaa asiakaspuolen (client-side) mallin. Uusien Java-widgettien tekeminen, sekä pelkästään asiakaspuolen sovellusten, eli ainoastaan selaimessa ajettavien sovellusten, kehittäminen tapahtuu tätä mallia hyödyntäen. Vaatimen asiakaspuolen sovelluskehys sisältää Google Web Toolkitin (GWT). GWT puolestaan sisältää kääntäjän, jolla saadaan kirjoitettu Java koodi käännettyä JavaScriptiksi, joka ajetaan selaimessa. GWT:stä löytyy myös täysin tuettu käyttöliittymäkehys. Näin ollen Vaatimen palvelinpuoli, sekä asiakaspuoli on puhdasta Javaa. (Vaadin Team 2017, 3.)

3.3 **Google Web Toolkit**

Google Web Toolkit (GWT) on Googlen kehitystyökalu, jota käytetään monimutkaisten selainpohjaisten sovellusten kehittämiseen ja optimointiin. Kyseessä on täysin ilmainen avoimen lähdekoodin työkalu, jota Google käyttää myös omissa tuotteissaan, kuten AdWordsissa ja Bloggerissa. GWT sisältää perusteellisen kokoelman Java Widgettejä ja Java API:ja, jotka se kääntää selaimelle sopivaan formaattiin, eli JavaScriptiin. Widgetillä tarkoitetaan käyttöliittymä komponenttia, joka on koodattu Javalla. Widgettiin on yleensä rakennettu myös toiminnollisuuksia, kuten eri toimintoja vuorovaikuttamaan käyttäjän kanssa. Esimerkiksi painike, jolla on valmiina onClick() -metodi. Java API:lla taas tarkoitetaan Java ohjelmointirajapintaa (Java application programming interface).

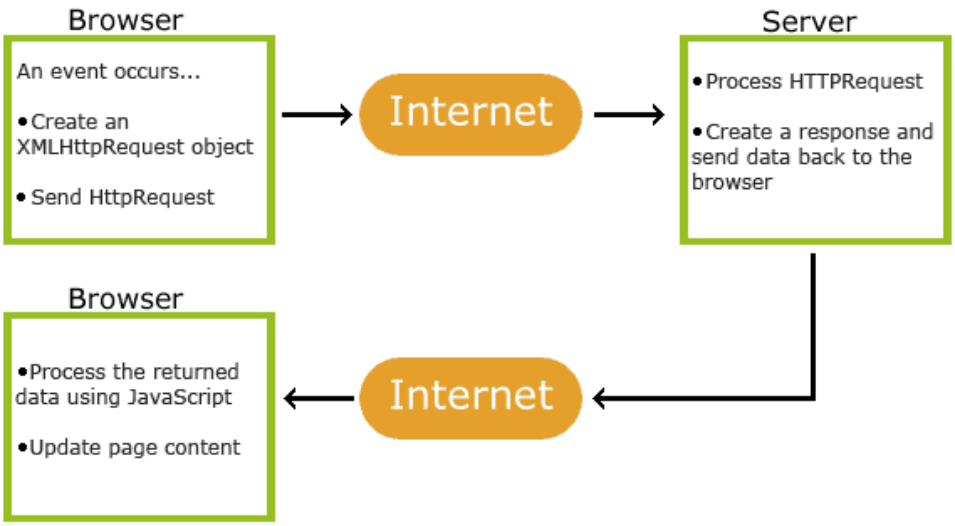
GWT mahdollistaa AJAX-sovellusten kirjoittamisen Javalla, minkä GWT kääntää optimoiduksi JavaScriptiksi, mikä toimii kaikissa selaimissa, sekä mobiiliselaimissa niin Androidissa, kuin iOS-laitteissakin. Vaadin käyttää GWT:tä selainpuolen renderöintimoottorin toteutuksessa, eli ohjelmoija voi halutessaan myös yhdistää Vaatimen GWT:hen. Kun sovellusta ajetaan,

eli sovellus on käynnissä, palvelinpuolella ei ole minkäänlaisia riippuvuuksia GWT:stä. Myös selaimessakin ajettava osa sovellusta on puhdasta JavaScriptiä ja riippumaton Googlesta. (GWT, n.d.) (Vaadin Team 2017, 62.)

Ohjelmoijan käyttäessä Vaadin sovelluskehystä, ei hänen tarvitse tietää mitään GWT:stä, tai ymmärtää miten selainpuoli toteutetaan sovelluksessa, ellei ohjelmoija halua tehdä uutta komponenttia, mitä ei olemassa olevien kirjastojen ja widgettien avulla voi tehdä. Tällöin komponentti luodaan kirjottaen laajennus Javalla vasten GWT:n ohjelmointirajapintaa, joka käännetään JavaScriptiksi käyttäen GWT:n käännintä. Eclipsen Vaadin-lisäosa ja Maven hoitavat tämän automaattisesti. (Vaadin Team 2017, 62.)

3.4 **AJAX**

AJAX on lyhenne seuraavasta termistä: ”Asynchronous JavaScript and XML”. AJAX:iä käytetään kehittämään web-sovellusten vuorovaikutteisuutta. AJAX:in avulla web-sovelluksista saadaan hyvinkin työpöytäsovelluksen kaltaisia. Sivustot ja sovellukset, jotka eivät hyödynnä AJAX-tekniikkaa, joutuisivat aina ladata sivun uudestaan tai ladata kokonaan uuden sivun, jos he haluaisivat näyttää uutta sisältöä.



Kuva 2. How AJAX Works (W3Schools, n.d.).

AJAX mahdollistaa uuden sisällön lataamisen ja saamisen palvelimelta, ilman, että tarvitsee ladata uutta sivua. AJAX-tekniikkaa toimii ylläolevan kuvan (Kuva 2.) mukaisesti. AJAX:ia hyödyntävän sivun vuorovaikutus käyttäjän kanssa hoituu JavaScriptillä.

Otetaan esimerkiksi tilanne, jossa käyttäjä on selaimella tietyllä sivulla, jossa on painettava näppäin, joka painaessaan näyttää uutta sisältöä sivulla, vaikka kolme uutta valokuvaa. Käyttäjä painaa sivulla olevaa näp-

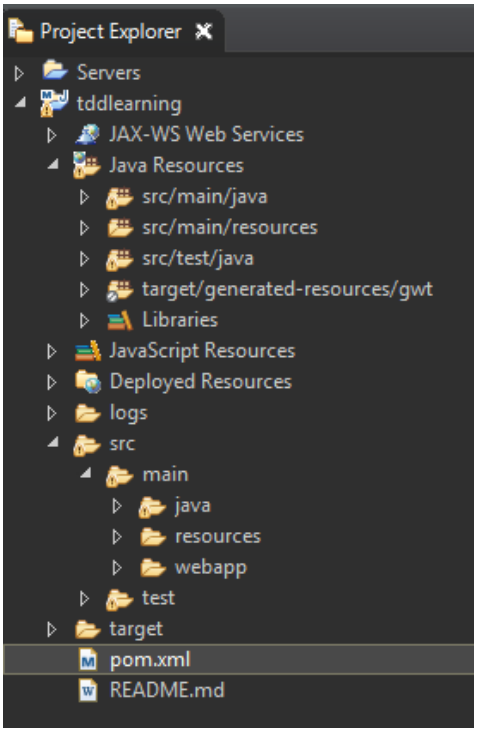
päintä, mikä ilmoittaa sivulle, että on tapahtunut tapahtuma (event). Tällöin sivusto luo JavaScripti-olion nimeltään XMLHttpRequest ja lähettää sen palvelimelle asynkronisesti. Palvelin käsittelee saadun pyynnön, luo vastauksen ja lähettää pyydetyt tiedot, tässä tapauksessa mm. ne kolme uutta valokuvaa, takaisin selaimelle. Selain käsittelee uuden datan hyödyntäen JavaScriptiä, ja muokkaa sivua datan tarpeiden mukaan. Hyödyntäen edellä mainittua tekniikkaa, tarvitsee vain pieniä osia sivusta ladata, jolloin kokonaan uutta sivua ei tarvitse ladata, ja sivuston käyttäytyminen on hyvin samankaltaista, kuin työpöytäsovelluksen.

AJAX ei ole yksi oma tekniikka, vaan kyseessä on joukko useita eri teknologioita, joilla web-sovelluksista tehdään vuorovaikutukseltaan enemmän työpöytäsovelluksien kaltaisia. Tämä saavutetaan yhdistelemällä useita tekniikoita; HTML, CSS, DOM, JavaScript sekä XMLHttpRequest-ohjelmointirajapintaa JavaScriptissä. Vaadin sovelluksessa, AJAX hoituu GWT:n kautta, joten ohjelmoijan ei tarvitse huolehtia, tai vaivata päätään asialla. XML on tietenkin vain yksi vaihtoehto siirtää dataa takaisin palvelimelta. Muita vaihtoehtoja ovat muun muassa HTML, JSON tai EBML. Vaadin sovelluksissa, XML:n käyttö datan serialisoinnissa, on korvattu sitä tehokkaammalla vaihtoehdolla, JSON:lla. (Vaadin Team 2017, 61.)

3.5 **Apache Maven**

Maven on Java-projekteissa käytettävä työkalu, jolla voidaan rakentaa ja hallita Java-projektia. Mavenin vahvuuksia ovat muun muassa automatisoitu kirjastojen hallinta, sekä Java-projektien siirrettävyys. Jokaisella Maven projektilla on pom.xml tiedosto, johon Maven projektit määritellään. POM on lyhenne sanasta project object model. Pom.xml tiedostosta löytyy muun muassa Maven projektin nimi ja omistaja, sekä kyseiseen tiedostoon määritellään myös tarvittavat kirjastot ja liitännäiset.

Maven pyrkii olemaan täysin omavarainen työkalu, sisällyttämällä muita yleisiä toiminnollisuuksia työkalupakkiinsa, kuten tarvittavien kirjastojen lataaminen ja asentaminen ilman ylimääräisiä skriptejä tai muita työkaluja. Kirjastojen lataaminen ja asentaminen tapahtuvat Mavenin kautta automaattisesti. Käyttäjän tarvitsee vain määritellä kirjastot pom.xml tiedostoon. Käyttäjän määritettyä kirjastot, Maven lataa kaikki käytettävät kirjastot, sekä niiden käyttämät kirjastot automaattisesti ja tallentaa ne Java-projektin välimuistiin. Sama toimintamalli tapahtuu eri liitännäisten lataamisessa ja asentamisessa.



Kuva 3. Mavenia käyttävän Java-projektin rakenne

Maven on saavuttanut erittäin vahvan suosion, ja onkin käytössä hyvin monessa työyhteisössä, sekä lukuisten harrastajien parissa. Vaihtoehtoisia työkaluja toki löytyy, joista mainitakseni muutamia ovat muun muassa Apachen Ant, sekä Gradle. (Maven, n.d.)

3.6 **Eclipse ja Vaadin-lisäosat**

3.6.1 Eclipse

Vaadin ei ole sidottu mihinkään tiettyyn ohjelmointiympäristöön. Halutessaan Vaadinta voi käyttää, vaikka ilman ohjelmointiympäristöä. Itse käytin kuitenkin ohjelmistoympäristönä Eclipseä. Eclipse on yksi suosituimmista ohjelmointiympäristöistä, kun puhutaan Java-kehityksestä. Eclipse on vapaan lähdekoodin ohjelmointiympäristö, jonka vahvuutena on eri liitännäiset. Eclipse tukee siis kolmannen osapuolen liitännäisiä. Nämä liitännäiset luovat toiminnallisuuden ja tekevät Eclipsestä sen, mitä se on: joustava ohjelmointiympäristö, johon voi itse tehdä, tai ladata liitännäisiä omien tarpeittensa mukaan. (Eclipse Foundation, n.d.).

Itse käytin ensimmäistä kertaa Eclipseä tämän projektin yhteydessä. Aiemmin kokemusta löytyi NetBeans:ta, sekä VisualStudio:sta. Tilaaja ehdotti Eclipsen käyttöä, sillä useat ohjelmistosuunnittelijatkin käyttävät kyseistä ympäristöä Triplan Oy:llä. Lisäosien ja liitännäisten lisäksi Eclipse tarjoaa helpotusta itse fyysiseen koodin kirjoittamiseen. Ympäristö tukee koodin

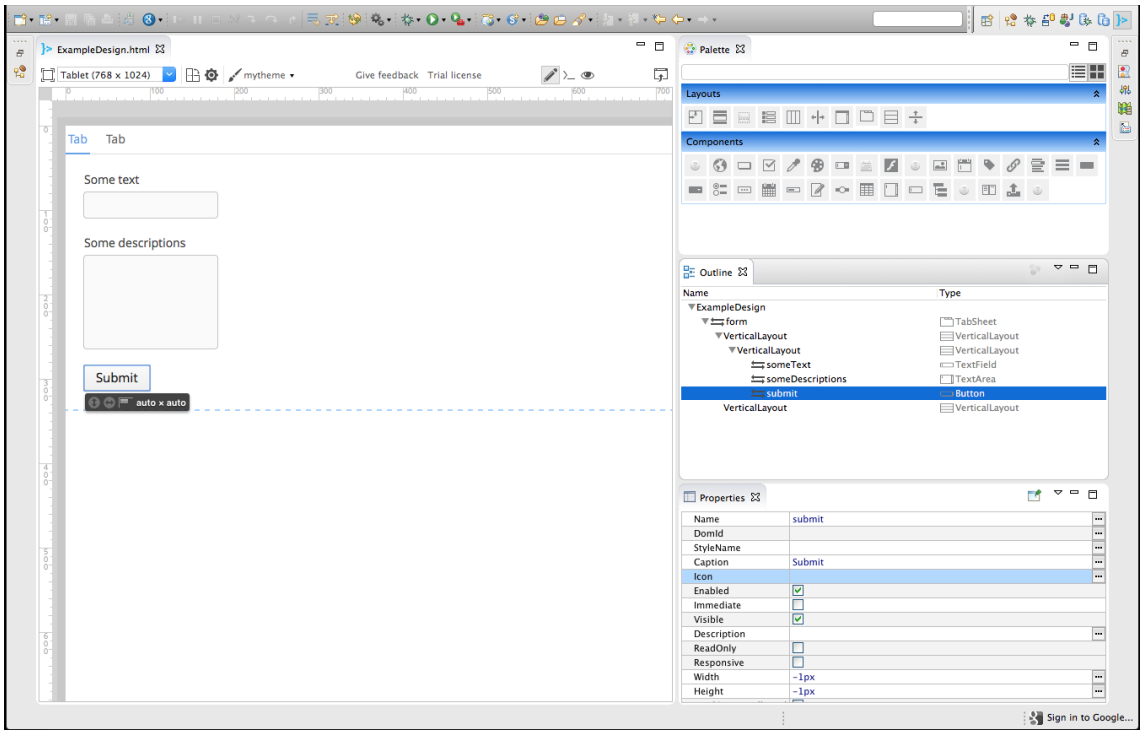
automaattista täydentämistä, tunnistaa virheet koodista ja tarjoaa niihin korjausehdotuksia. Koodin jäsentäminenkin onnistuu kätevästi vain valitsemalla valikosta kyseinen kohta. Eclipse tarjoaa myös lukuisia käytännöllisiä toimintoja muun muassa refaktorointiin liittyen.

3.6.2 Vaadin IDE

Valitsin ohjelmointiympäristökseni Eclipsen, sillä Vaadin tarjoaa Vaatimen oman liitännäisen, Vaadin IDE:n muutamalle suositulle Java-kehitysalustalle, joista yksi on Eclipse. Vaadin IDE on myös saatavilla Netbeans:lle sekä IntelliJ IDEA:lle. Kyseinen liitännäinen auttaa uusien Vaadin projektien luomisessa, sekä mukautettujen teemojen ja widgettien luomisessa. Liitännäinen tekee myös Vaadin kirjaston päivittämisestä helpompaa, joten uudemman Vaadin kirjaston päivittäminen vanhemmissakin projekteissa olisi mahdollista ja helpompaa. (Vaadin Team 2017, 5.)

3.6.3 Vaadin Designer

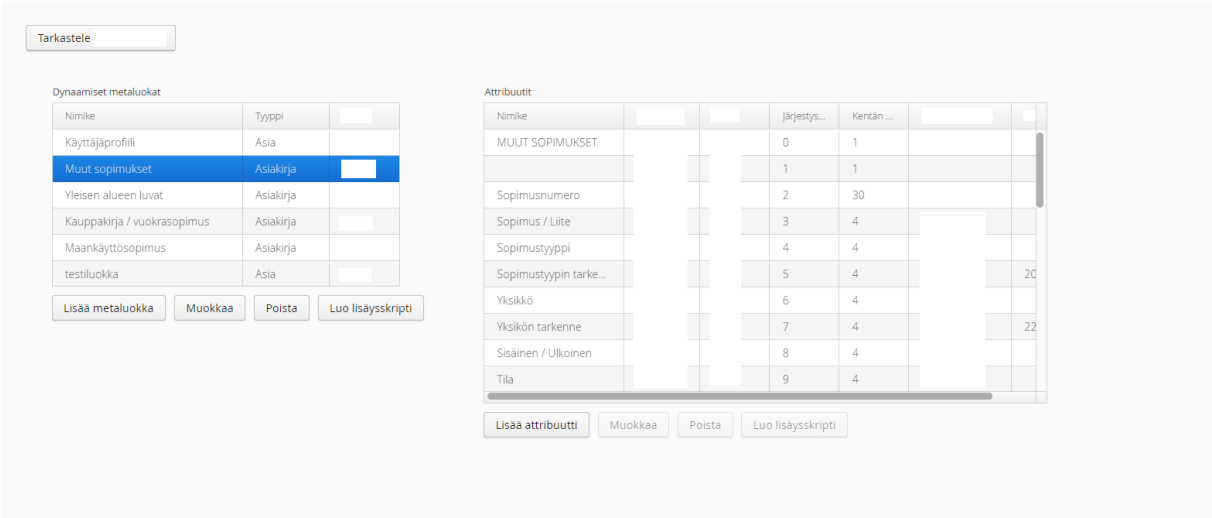
Vaadin Designer (Kuva 4.) on Vaatimen kaupallinen liitännäinen. Kyseisellä liitännäisellä käyttöliittymäsuunnittelu, sekä toteuttaminen on huomattavasti tehokkaampaa, kuin että kirjoittaisi käsin aina visuaaliset komponentit. Kyseessä on siis visuaalinen suunnittelutyökalu ammattikehittäjille, joka toimii raahaa ja tiputa -periaatteella. Vaadin Designer toimii hyvin myös luonnostelutyökaluna, kuin myös tapana opiskella ja tutustua Vaatimen eri komponentteihin ja ulkoasuihin eli layoutteihin. (Vaadin Team 2017, 267.)



Kuva 4. Vaadin Designer Views (Vaadin, 2017)

4 DYNAAMISIA METATIETOJA HALLINNOIVA SOVELLUS

Perehtyessäni Vaadin sovelluskehykseen sekä Twebin metatietojen rakenteeseen pidin seuraavaa kysymystä hyvin tärkeänä: Mitä tietoa tulisi käyttäjälle näyttää ja miten ne näytetään? Hahmotettuani Twebin metatietojen rakenteen, sekä Vaadin sovelluskehyksen perusteet, pidimme muutaman tapaamisen Triplanin tiloissa. Tapaamisissa suunniteltiin ja ideoitiin kyseistä sovellusta ja näistä oli todella suuri apu sovelluksen suunnittelussa. Rajasimme alkuun sovelluksen perustoiminnot ja visio siitä, miltä se voisi mahdollisesti näyttää, alkoi hahmottua (Kuva 5.).



Kuva 5. Hahmotelma mahdollisesta sovelluksen päänäkymästä

Suunnitteluvaiheen alkupuolella olin jo vakuuttunut siitä, että sovelluksessa tulisi olemaan 2 taulu-elementtiä, joissa näytettäisiin dynaamiset metaluokat, sekä niihin sidotut attribuutit. Tämä säilyi läpi sovellussuunnittelun ja -toteutuksen perusideana.

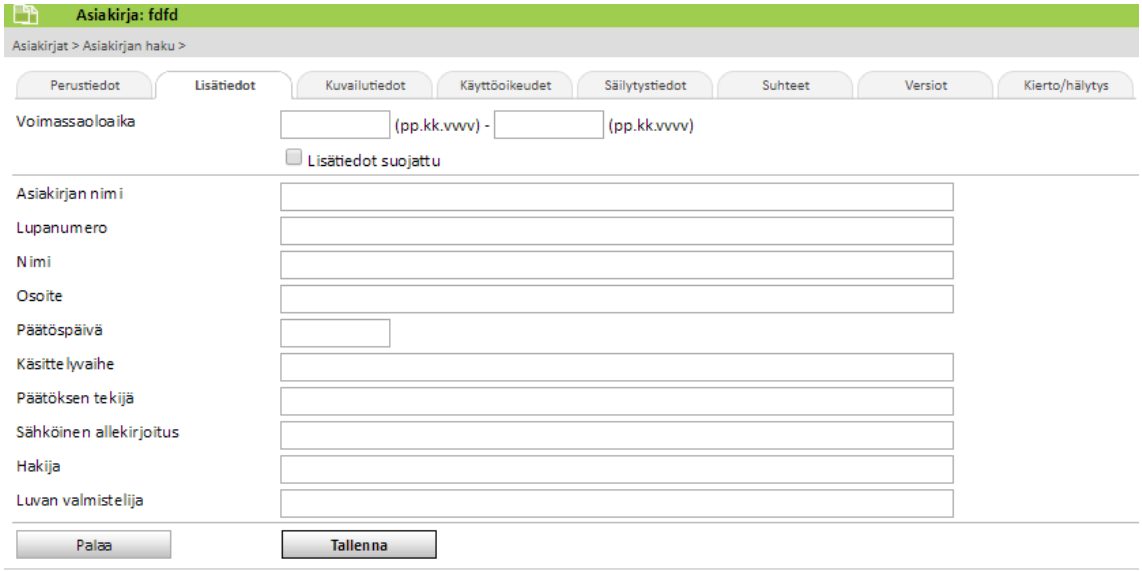
Sovelluksen kehittämisessä hyödynsin prototyyppimenetelmää. Prototyyppimenetelmässä rakennetaan käyttöliittymä ensin, ja tämän jälkeen lisätään liiketoiminnallinen kerros sekä tietokantakerros. Näiden vaiheiden jälkeen on niin sanottu kehityskierros saatettu loppuun, jonka jälkeen voidaan syventää ja laajentaa suunnittelua. Prototyyppimenetelmän etuna on sovelluksen nopea havainnollistaminen asiakkaalle, sillä kun luodaan käyttöliittymä ensin, voidaan tarvittavia muutoksia tehdä vielä suhteellisen helposti ja ennen kaikkea kustannustehokkaasti.

Valitsin prototyyppimenetelmän siksi, että oma kokemukseni sovellussuunnittelusta oli vähäistä, ja sovelluksen vaatimukset elivät vielä. Rakennettuani käyttöliittymää pala palalta ja esiteltyäni sitä ohjaajalle varmistin, että olen samalla sivulla tilaajan kanssa siitä, millaiselta sovellus voisi mahdollisesti näyttää. Käyttöliittymän jälkeen oli toiminnallisuutta helppo lisätä samalla periaatteella; pala palalta, varmistaen sen toimivuus ja käytännöllisyys. (Wikipedia 2017 b.)

4.1 Dynaamisten metatietojen rakenne

Ennen kuin pureudun itse sovelluksen suunnitteluun ja ohjelmointiin, avaan ensin dynaamista metatietoa käsitteenä, sekä sen rakennetta. Käsitteen ymmärtäminen on hyvin tärkeää, sillä sovellushan luotiin juuri dynaamisten metatietojen hallinnoimisen apuvälineeksi, helpottamaan prosessia. Metatiedollahan tarkoitetaan kuvailevaa tietoa suuremmasta tietovarannosta, niin kuin aikaisemmin Johdanto-osiossa käy ilmi.

Triplanin päätuotteessa, Twebissä, dynaamiset metatiedot ilmenevät peruskäyttäjälle Lisätiedot-välilehdessä olevien kenttien muodossa (Kuva 6.).



Kuva 6. Esimerkkiasiakirjan Lisätiedot-välilehti Tweb:ssä.

Yllä olevassa kuvassa (Kuva 6.) on luotu tyhjä asiakirja mielivaltaisella nimellä. Asiakirjalle on määritetty dynaamiseksi metaluokaksi eräs metaluokka, jolla on useita attribuutteja. Nämä attribuutit ovat Lisätiedot-välilehdessä sijaitsevat kentät. Kyseiset kentät ovat metaluokan attribuutteja. Attribuuteilla voidaan siis kuvata ja määrittää tietoa, tässä tapauksessa asiakirjaa, tarkemmin täydentämällä kenttiin tietoa.

Dynaamiset metatiedot koostuvat siis asiakirjalle ja/tai asialle määritettävästä metaluokasta, joka pitää sisällään aina asiakirjasta ja/tai asiasta, sekä

asiakkaasta riippuen erinäisiä attribuutteja, eli kenttiä. Attribuuteille voidaan myös määrittää, ovatko ne tekstikenttiä, päivämääräkenttiä, valintalistoja tai valintaruutuja. Valintalistalle pitää luoda valintalista-arvoja, joista käyttäjä valitsee yhden arvon. Jokaiselle valintalistalle luodaan omat valintalista-arvot.

Metaluokat, attribuutit ja valintalista-arvot eritellään toisistaan relaatiotietokannassa yksilöivien avaimien avulla. Tässä tapauksessa kyseessä on numeerinen arvo, joka on nimeltään id. Twebin relaatiotietokannassa on metaluokille omat taulunsa, sekä attribuuteille omansa. Attribuutit yhdistetään metaluokkaan käyttämällä sidostaulua, missä määritellään ristiviittauksella, mille metaluokalle mikäkin attribuutti kuuluu. Metaluokilla, attribuuteilla, sekä valintalista-arvoilla on myös muita useita arvoja, tai kolumneja, jos tarkastellaan asiaa relaatiotietokannan näkökulmasta.

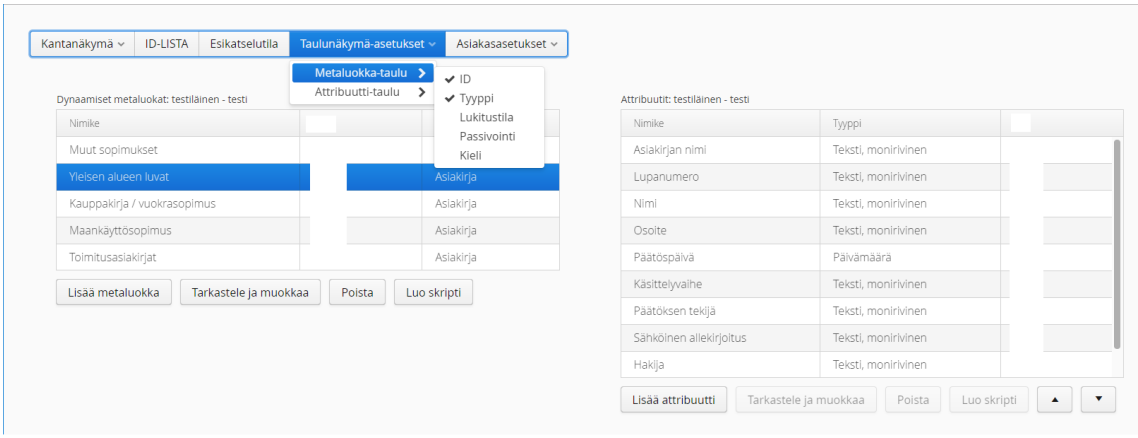
4.2 **Sovelluksen käyttöliittymä**

Käyttöliittymälle asetetut odotukset olivat selkeät ja yksimieliset: helppokäyttöinen ja selkeä. Sovelluksen käyttäjän tulisi pystyä käyttämään sovellusta ilman, että hänen tarvitsisi tietää, miten metaluokkia ja niiden attribuutteja luodaan tietokantatasolla. Käyttäjän ei tarvitsisi huolehtia päällekkäisyyksistä, yksilöivistä avaimista, skripteistä tai metaluokkien dokumentoinnista. Sovellusta tulisi pystyä käyttämään pelkästään sillä tiedolla, minkä sovellusasiantuntija sopii asiakkaan kanssa. Käyttäjä sopii siis asiakkaan kanssa ennakkoon dynaamisten metaluokkien määrittelyn, missä tarkennetaan metaluokkien, sekä attribuuttien erinäisiä arvoja, joita ovat muun muassa kieli, nimi ja datatyyppi. Sovellus hoitaisi loput joko automaattisesti tai käyttäjää avustaen.

4.2.1 Sovelluksen päänäkymä

Päädyin ratkaisuun, jossa sovelluksen päänäkymässä olisi 2 taulua. Näissä tauluissa näytettäisiin metaluokat, sekä siihen sidotut attribuutit, niin kuin kuvassa (Kuva 5.) on toteutettu. Tauluina käytin Vaadin sovelluskehyksen Grid-komponenttia. Oletettavasti, käyttäjä ei halua nähdä mitään ylimääräistä metaluokkiin ja attribuutteihin liittyen. Näin pystytään pitämään käyttöliittymä ja ylipäätänsä sovelluksen käyttökokemus helppona ja selkeänä. Tämän toteutin näyttämällä tauluissa vain metaluokkien ja attribuuttien nimet, id:t, sekä datatyypin.

Joustavuus on hyvin tärkeä asia suunnitellessa käyttöliittymää. Joustavuudella tarkoitan toimintoja, joilla käyttäjä voi muokata käyttöliittymää esimerkiksi näyttämään enemmän tietoa, tai vaihtoehtoisesti vähentämään tiedon määrää, omalle silmälle sopivaksi. Sovellukselle saadaan lisää joustavuutta, kun käyttäjälle luodaan mahdollisuus muokata asetuksia. Esimerkiksi edellä mainittujen metaluokka- ja attribuuttitaulun tiedonmäärän näkyvyyteen liittyen, lisäsin käyttäjälle mahdollisuuden lisätä tai vähentää molempien taulujen näkyvää tietomäärää (Kuva 7.).

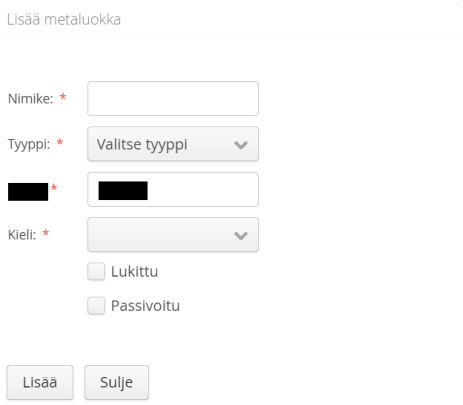


Kuva 7. Taulunäkymä-asetuksien muuttaminen sovelluksessa

Kyseinen toiminto mahdollistaa metaluokkien ja attribuuttien yksityiskohtaisemman tarkastelun, kun tarve vaatii, tai kun käyttäjä haluaa muuten vain nähdä enemmän dataa niihin liittyen.

4.2.2 Dynaamisten metatietojen käsittelynäkymät

Dynaamisten metatietojen lisääminen ja muokkaaminen tapahtuvat päänäkymällä sijaitsevien painikkeiden kautta. Kun käyttäjä haluaa lisätä metaluokan tai attribuutin, pystyy hän tehdä sen painamalla joko ”Lisää metaluokka”-näppäintä, tai ”Lisää attribuutti”-näppäintä. Kyseiset näppäimet sijaitsevat taulujen alla ensimmäisinä näppäinriveissään. Painaessaan ”Lisää metaluokka”-näppäintä, aukeaa käyttäjälle kuvan mukainen näkymä (Kuva 8.).

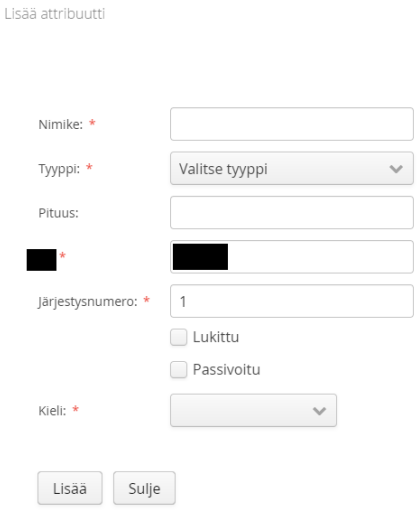


Kuva 8. Metaluokan lisäämisnäkymä

Näkymä koostuu täytettävistä tekstikentistä, valintalistoista, sekä valintaruuduista. Sovellus hoitaa kaiken vaativamman tiedon käsittelyn ja syöttämisen. Käyttäjän tulee syöttää metaluokalle nimi, sekä valita metaluokan tyyppi valintalistasta. Tämän jälkeen pitää vielä valita metaluokalle kieli, sekä määrittää onko kyseessä mahdollisesti lukittu tai passivoitu metaluokka. Sovellus huolehtii itse validin id-arvon syöttämisestä, ja esitäyttää sen käyttäjän puolesta.

Vastaavasti, kun käyttäjä haluaa muokata tai katsoa tarkemmin metaluokkaa, valitsee hän metaluokan taulusta ja painaa taulun alla olevaa ”Tarkastele ja muokkaa”-painiketta. Tällöin aukeaa täysin samanlainen näkymä, kuin luodessa uutta metaluokkaa (Kuva 8.), mutta kaikki tiedot ovat esitäytettyjä kyseisen metaluokan tietojen mukaan. Käyttäjä muokkaa vaihtamansa tiedot, ja painaa ”Päivitä”-näppäintä, jonka jälkeen kyseisen metaluokan muutetut tiedot päivittyvät sovelluksen kantaan.

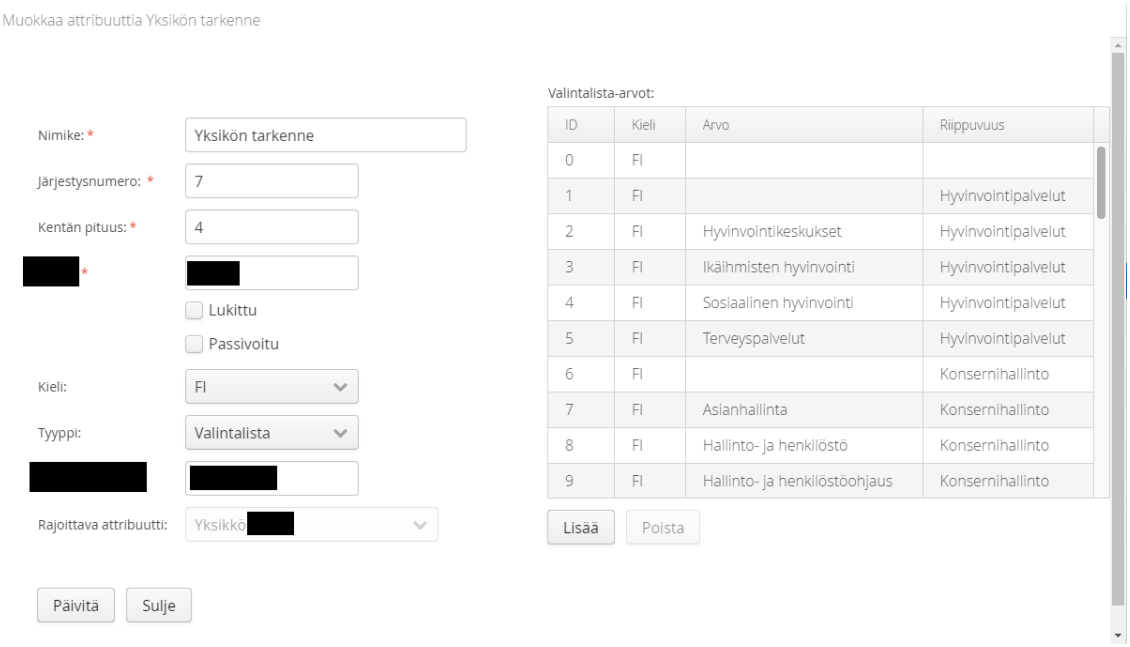
Lisätäkseen attribuutin, tulee käyttäjän ensin valita metaluokka, mihin attribuutti tullaan lisäämään. Valittuaan metaluokan, pystyy käyttäjä painamaan attribuutti-taulun alla sijaitsevaa ”Lisää attribuutti”-näppäintä. Painettuaan kyseistä näppäintä, aukeaa käyttäjälle kuvan mukainen näkymä (Kuva 9.).



Kuva 9. Lisää attribuutti-näkymä

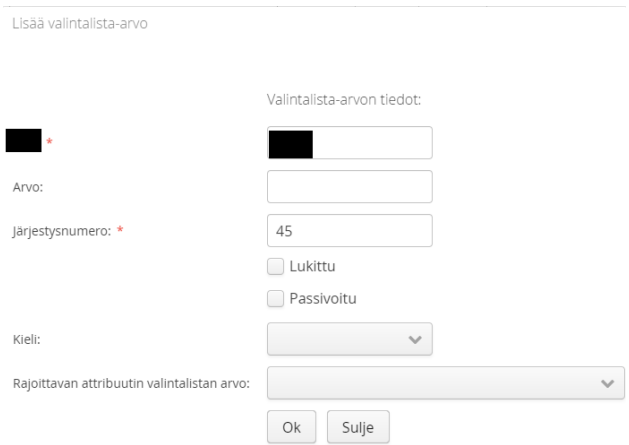
Näkymä on hyvin samankaltainen, kuin metaluokkaakin lisätessä. Muutamia poikkeuksia kuitenkin löytyy, kuten pituus- ja järjestysnumerokenttä. Riippuen attribuutin tyypistä, sovellus lisää syötettäviä kenttiä lisää, esimerkiksi jos tyypiksi valitaan valintalista, tulee attribuutille antaa valintalistalle spesifioituja arvoja. Käyttäjän syötettyä kaikki tarvittavat tiedot, ja painettua ”Lisää”-näppäintä, lisää sovellus attribuutin tietokantaansa.

Attribuutin ”Tarkastele ja muokkaa”-näkymä on myös hyvin samankaltainen, kuin metaluokan vastaava näkymä, paitsi kun attribuutin tyyppi on valintalista. Tällöin pitää pystyä näyttämään ja käsittelemään valintalista-arvot, jolloin ”Tarkastele ja muokkaa”-näkymään lisätään taulukko, jossa näytetään kyseisen valintalistan arvot. Kyseisen taulukon alta löytyy myös painikkeet, joilla valintalista-arvoja voidaan poistaa tai luoda uusia (Kuva 10.).



Kuva 10. Attribuutin "Tarkastele ja muokkaa"-näkymä kun attribuutin tyyppinä on valintalista.

Näkymä toimii täysin samalla logiikalla kuin metaluokan vastaava näkymä. Käyttäjä tekee tarvittavat muutokset valitsemaansa attribuuttiin ja painamalla ”Päivitä”-näppäintä tiedot päivittyvät suoraan sovelluksen kantaan. Käsitellessä valintalistoja, voi käyttäjä halutessaan lisätä valintalista-arvoja painamalla ”Lisää”-näppäintä. Painallus avaa uuden näkymän, jossa luodaan valintalista-arvo kyseiselle valintalistalle, eli attribuutille (Kuva 11.).



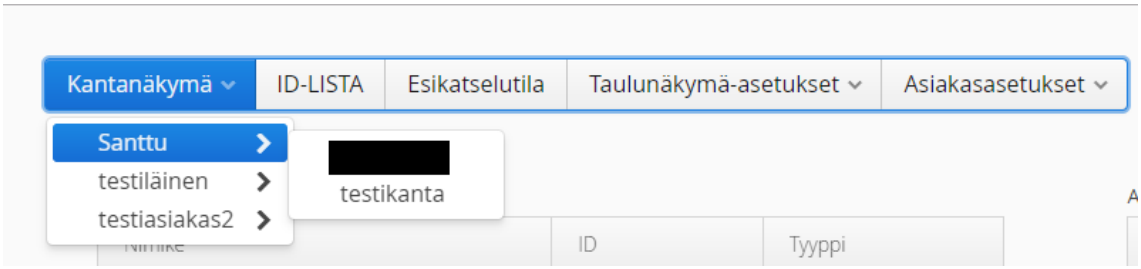
Kuva 11. Valintalista-arvon lisäys-näkymä.

Sovellus esitäyttää jälleen id-arvon sekä järjestysnumeron automaattisesti, niin kuin attribuuttiakin luodessa. Käyttäjä syöttää tarvittavat tiedot ja painaa näkymän alhaalla sijaitsevaa ”Ok”-näppäintä. Tällöin valintalista-arvo

lisätään sovelluksen tietokantaan ja siihen sidottuun attribuuttiin, sekä näkymä sulkeutuu ja palaa attribuutin ”Tarkastele ja muokkaa”-näkymään.

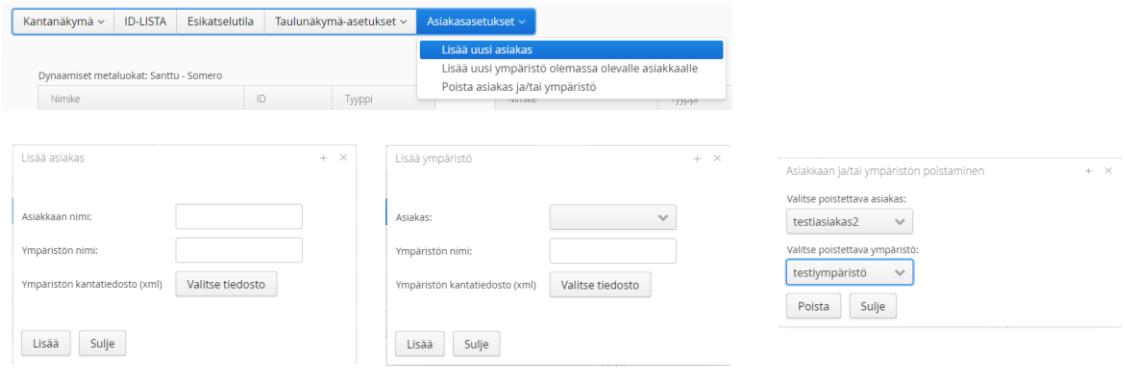
4.2.3 Navigaatiopalkki

Kuten kuvasta (Kuva 7.) käy ilmi, päätin lisätä myös sovelluksen päänäkymään taulujen yläpuolelle komponentin, josta käyttäjä voi vaihtaa asetuksia ja navigoida päänäkymästä sovelluksen muihin näkymiin. Koin tämän tarpeelliseksi, sillä sovellus tulee sisältämään useamman asiakkaan dynaamiset metatiedot, jotka voivat vaihdella hyvinkin suuresti keskenään. Myös sovelluksen laaja määrä erilaisia toimintoja vaati navigaatiopalkin lisäämistä, sillä asetusten ja näkymien vaihtaminen on käyttöliittymän kannalta selkeämpää ja helpompaa, kun on navigaatiopalkin kaltainen komponentti. Sovellusta käyttäessä, käyttäjä haluaa tietenkin nähdä vain sen hetkisen asiakkaan dynaamiset metatiedot, ja ehkä jopa tietyn ympäristön dynaamiset metatiedot. Sovelluksessa tämä toteutettaisiin mahdollisuutena vaihtaa kantanäkymää.
Taulujen yläpuolella olevasta navigaatiopalkista ”Kantanäkymä”-kohdasta käyttäjä saisi alasveto-valikon (Kuva 12.) avulla valita haluamansa asiakkaan sekä asiakkaan ympäristön, joka alla olevissa tauluissa näkyisi.



Kuva 12. Asiakkaan ja ympäristön valinta navigaatiopalkista sovelluksessa

Lisätäkseen tai poistaakseen asiakkaita ja heidän ympäristöjään, on navigaatiopalkissa kohta ”Asiakasasetukset”. Klikkaamalla kyseistä kohtaa navigaatiopalkissa, aukeaa käyttäjälle alasvetovalintalista, josta voi valita haluaako lisätä kokonaan uuden asiakkaan, uuden ympäristön olemassa olevalle asiakkaalle, tai poistaa asiakkaita ja ympäristöjä (Kuva 13.).



Kuva 13. Asiakasasetuksien valintalista-arvot, sekä niistä aukeavat uudet näkymät.

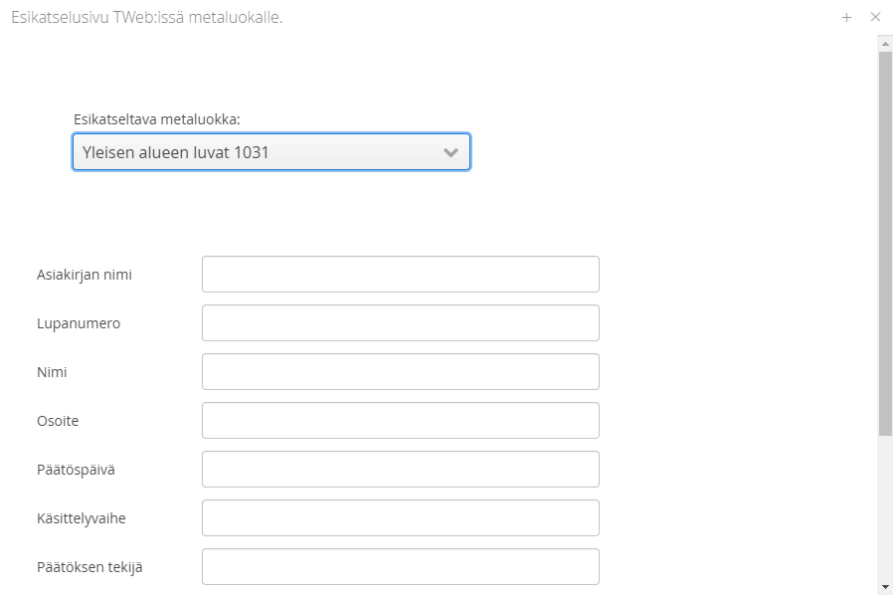
Sovelluksen käyttäminen aloitetaan, lähes tulkoon aina, joko lisäämällä uusi asiakas tai ympäristö. Tämä johtuu siitä, että käyttäjän aloittaessa työtehtäväänsä, tulee hänellä olla asiakkaan viimeisin tietokantarakenne käytössä. Tästä syystä, käyttäjä joko luo aina uuden asiakkaan, tai ympäristön, missä asiakkaan vaatimat lisäykset ja muutokset tehdään.

Käyttäjän halutessa lisätä uusi asiakas, hän painaa Asiakasasetusten alasvetovalintalistasta kohtaa ”Lisää uusi asiakas”. Tästä aukeaa näkymä, joka on kuvan (Kuva 13.) vasemman puoleisin ikkuna: ”Lisää asiakas”. Kyseisessä näkymässä käyttäjä täyttää tarvittavat tiedot, sekä lataa palvelimelle asiakkaan kantatiedoston. Käyttäjä saa asiakkaan kantatiedoston käyttämällä tekemääni komentoriviltä ajettavaa Java-sovellusta. Tämä komentorivisovellus muodostaa asiakkaan sen hetkisestä kannasta xml-tiedoston. Painaessaan ”Lisää” sovellus luo uuden asiakkaan ja ympäristön, jonka tietokanta vastaa sovellukseen ladatun xml-tiedoston kantarakennetta.

Uuden ympäristön lisääminen jo olemassa olevalle asiakkaalle noudattaa hyvin pitkälti samaa protokollaa, kuin uuden asiakkaankin. Ainoana erona on, että asiakas valitaan valintalistasta, eikä sille anneta nimeä, niin kuin uutta asiakasta luodessa. Uuden ympäristön lisäämisen näkymä on nähtävissä kuvan (Kuva 13.) keskimmäisimmässä ikkunassa.

Käyttäjän halutessa poistaa asiakas tai ympäristö sovelluksesta, valitsee hän alasvetovalikosta kohdan ”Poista asiakas ja/tai ympäristö”. Tällöin aukeaa näkymä, joka on kuvan (Kuva 13.) oikeanpuoleisin ikkuna. Näkymä sisältää kaksi valintalistaa, joista ensimmäisestä valitaan poistettava asiakas ja toisesta poistettava ympäristö. Jälkimmäisestä valintalistasta on oletuksena valittu arvo ”Kaikki”. Tällöin sovellus poistaa asiakkaan ja kaikki sen ympäristöt tietokannasta. Vastaavasti, jos käyttäjä haluaa poistaa, jonkun tietyn ympäristön asiakkaalta, valitsee hän silloin valintalistasta tämän kyseisen ympäristön, ”Kaikki”-arvon sijaan.

Navigaatiopalkki sisältää, yllä olevien ominaisuuksien ja toiminnollisuuksien lisäksi, myös mahdollisuuden siirtyä esikatselutilaan. Tämä onnistuu painamalla navigaatiopalkista kohtaa ”Esikatselutila”. Esikatselutila esittää metaluokan attribuutit sellaisenaan, kuin ne näkyisivät Twebissä. Esikatselutilan tarkoitus on antaa visuaalista palautetta käyttäjälle tekemästään työstä, oli se sitten uusien dynaamisten metatietojen luomista tai vanhojen muuttamista. Tässä näkymässä (Kuva 14.) käyttäjä saa varmuuden siitä, miltä jälki näyttää, ja ennen kaikkea saa varmuuden siitä, että ei ole rikkonut mitään. Esikatselutila antaa myös loistavan mahdollisuuden varmistaa asiakkaalta, että tekemä työ on se, mitä asiakas vaati. On huomattavasti helpompaa lähettää näyttökaappaus esikatselutilasta asiakkaalle varmistukseksi, kuin ajaa luodut skriptit testiympäristöön ja pyytää asiakkaan tarkistaa, että näyttääkö nyt hyvältä.



Kuva 14. Esikatselutila sovelluksessa.

4.3 Sovelluksen liiketoiminnallinen kerros

Sovelluksen liiketoiminnallinen kerros, eli niin sanottu bisnes-logiikka on se, mikä oikeuttaa sovelluksen olemassaolon. Se on sovelluksen punainen lanka. Liiketoiminnallisessa kerroksessa ilmenee sovelluksen logiikka, joka takaa toiminnollisuuden. Ilman bisnes-logiikkaa, sovellus olisi vain pelkkä käyttöliittymä ilman mitään toiminnollisuutta. Liiketoiminnallisessa kerroksessa toteutetaan siis sovellukselle määritetyt vaatimukset ja toiminnot.

Sovelluksen määritetyt toiminnot ja vaatimukset rajattiin sovelluksen suunnitteluvaiheessa. Sovelluksen tavoitteena oli helpottaa dynaamisten metatietojen hallinnointia. Päästä eroon vanhasta tavasta, jossa dynaamiset metatiedot luotiin skripteillä, mitkä saatiin hyödyntäen Excel-taulukkopohjia. Kyseinen tapa luoda dynaamisia metatietoja oli työlästä ja vaikeaa pitää ajan tasalla, sillä Excel-taulukoita oli useita ja niiden ylläpitäminen oli työlästä. Toiminnollisuuksiltaan sovelluksessa pitäisi siis pystyä luomaan, poistamaan ja muokkaaman dynaamisia metaluokkia, sekä luoda näistä tarvittavat skriptit. Dynaamisten metatietojen hallinnoiminen toteutettaisiin sovelluksen sisällä olevalla listalla, josta käyttäjä voi tarkastella asiakkaiden ja niiden ympäristöjen dynaamisia metatietoja.

4.3.1 Asiakkaan ja asiakkaan ympäristön tuominen sovellukseen

Sovelluksen käyttöliittymää suunnitellessa käytin tietokantapohjana Twebin tietokantaa. Sovelluksen tulisi kuitenkin tukea useampaa asiakasta, sekä heidän ympäristöjä. Tästä johtuen sovellukselle tehtiin oma tietokantarakenne, muokaten käyttämääni Twebin tietokantapohjaa niin, että olisi mahdollista esittää samassa tietokannassa useita asiakkaita, sekä heidän ympäristöjä. Asiakkaalla voi helposti olla useita ympäristöjä, joiden dynaamisia metatietoja pitää pystyä hallitsemaan, esimerkiksi tuotanto-, testi-, sekä koulutusympäristö.

Suurin huolenaihe tällaisessa tilanteessa, jossa säilytetään asiakkaiden dynaamisia metatietoja yhdessä sovelluksessa, on tietenkin se, että miten voidaan varmistua siitä, että sovelluksessa olevat dynaamiset metatiedot ovat ajan tasalla. Tähän ratkaisuna kehitin java-sovelluksen, jota ajetaan komentoriviltä. Kyseinen sovellus luo XML-tiedoston asiakkaan kannasta, joka tuodaan käyttöliittymän kautta dynaamisia metatietoja hallinoivaan sovellukseen. Dynaamisia metatietoja hallinoiva sovellus käsittelee XML-tiedoston, ja lisää asiakkaan tietokannan dynaamiset metaluokat omaan kantaansa. Käyttäjä voi luoda joko uuden asiakkaan, sekä ympäristön, tai luoda vain uuden ympäristön jo olemassa olevalle asiakkaalle, sekä tietenkin poistaa asiakkaita ja heidän ympäristöjään.

Tämä takaa prosessin, jolla varmistutaan siitä, että kyseinen kanta, joka sovelluksessa näkyy, on ajan tasalla. Kuvitellaan tilanne, jossa käyttäjä saa tehtäväkseen luoda uusia dynaamisia metaluokkia asiakkaalle. Ennen kyseisten luokkien luomista sovelluksella, tulee hänen, tai vaihtoehtoisesti asiakkaan, ajaa komentorivisovellus, jotta saadaan viimeisin tietokanta asiakkaalta. Tämän jälkeen käyttäjä vie tiedoston sovellukseen, jonka jälkeen hänellä on viimeisin kanta sovelluksessa, ja tekee tarvittavat muutokset. Viimeisenä, hän luo skriptit sovelluksella ja toimittaa ne eteenpäin. Tilanteessa, jossa kannassa on vanhempi versio asiakkaan kannasta, tulee käyttäjän poistaa vanha kanta sovelluksesta. Ongelmatilanteiden sattuessa, käyttäjä voi aina poistaa ympäristön, ja ladata kannan uudestaan, sillä palvelimella säilyy käyttäjän tuoma xml-tiedosto.

4.3.2 Dynaamisten metatietojen esittäminen ja hallinnointi sovelluksessa

Sovellus esittää dynaamiset metatiedot päänäkymässä kahden taulun avulla. Taulut esittävät kahdessa eri listassa olevat objektit, jotka tässä tapauksessa ovat metaluokat ja niihin sidotut attribuutit. Listat saavat sisältönsä hyödyntäen sovelluksen yhteyttä relaatiotietokantaansa. Sovellus suorittaa SQL-hakuja, joissa valitaan valitun asiakkaan ja ympäristön dynaamiset metaluokat, sekä niihin sidotut attribuutit. Näistä SQL-hauista muodostetaan uusi Java-olio, jokaista metaluokkaa ja attribuuttia kohden, mitkä lisätään omalle listalleen.

Dynaamisten metatietojen hallinnoiminen onnistuu sovelluksessa päänäkymän kautta. Käyttäjä valitsee taulusta haluamansa metaluokan tai attribuutin. Taulujen alla sijaitsevilla painikkeilla käyttäjä voi halutessaan poistaa, muokata tai generoida skriptit valitsemastaan metaluokasta tai attribuutista. Taulujen alla sijaitsevat myös painikkeet ”Lisää metaluokka”, sekä ”Lisää attribuutti”, joilla käyttäjä voi lisätä kyseisen asiakkaan ympäristöön metaluokan tai attribuutin.

Käyttäjän lisätessä, eli luodessa, uutta metaluokkaa tai attribuuttia, sovellus ehdottaa automaattisesti seuraavaksi vapaana olevan id-arvon uudelle metaluokalle tai attribuutille. Kyseinen toiminto perustuu siihen, että luodaan kaksi listaa, joissa ensimmäisessä listassa on määritellyn id-arvon ensimmäinen arvo, sekä kaikki arvot kyseisen määrityksen viimeiseen arvoon. Toiseen listaan haetaan nyt jo käytössä olevat id-arvot kyseisen asiakkaan ympäristöstä SQL-kyselyllä. Tämän jälkeen poistetaan ensimmäisestä listasta kaikki toisen listan arvot. Näin ensimmäisessä listassa on vain niitä lukuja, joita voidaan käyttää metaluokan tai attribuutin luomisvaiheessa. Sovellus esitäyttää id-kentän listan ensimmäisellä arvolla. Tällöin sovelluksen käyttäjän ei tarvitse itse miettiä, että mikä id on käytössä ja mikä ei.

Sovellus tarkistaa asiakkaan syötteen, ja esitäyttää muun muassa järjestysnumeron id-arvon lisäksi. Tämän ansiosta uusien metaluokkien ja attribuuttien luominen on vaivatonta, sekä vältytään ongelmatilanteista, missä kahdella attribuutilla tai metaluokalla olisi sama järjestysnumero tai id-arvo.

Kun käyttäjä on täyttänyt tarvittavat kentät lisätessä metaluokkaa tai attribuuttia ja painaa käyttöliittymästä ”Lisää”-näppäintä, tarkistaa vielä sovellus, että kaikki tarvittava tieto on täytetty. ”Lisää”-näppäintä painaessa ajaa sovellus omaan relaatiotietokantaansa juuri syötetyt arvot, ja luo kantaan kyseisen attribuutin tai metaluokan. Ennen kuin sovellus ajaa luodut arvot SQL-lauseina kantaan, varmistaa sovellus, että jokainen ajettava lause menee virheettömästi läpi. Virheen ilmetessä sovellus ei aja yhtäkään lausetta tietokantaan, vaan palauttaa käyttäjälle virheilmoituksen. Tällä varmistutaan, että tietokanta pysyy ehjänä, eikä päästä niin sanottua rikkinäistä tietoa sisään.

Käyttäjän muokatessa jo olemassa olevia metaluokkia tai attribuutteja, sovellus päivittää muokatut tiedot suoraan sovelluksen kantaan. Tässäkin tapauksessa päivitetyt tiedot tarkistetaan ennen niiden kantaan ajamista. Virheen ilmetessä mikään päivityslause ei mene läpi, vaan sovellus palauttaa käyttäjälle virheilmoituksen.

Halutessaan poistaa dynaamisia metatietoja, käyttäjä valitsee taulusta haluamansa metaluokan tai attribuutin ja painaa valitsemansa taulun alapuolelta löytyvää ”Poista”-painiketta. Poistaessa metaluokkaa, sovellus pois-

taa myös siihen liitetyt attribuutit, sekä mahdolliset attribuuttien valintalista-arvot. Attribuuttia poistaessa, sovellus poistaa vain kyseisen attribuutin, sekä siihen sidotut mahdolliset valintalista-arvot. Kun dynaaminen metatieto poistetaan sovelluksella, poistuu se suoraan myös sovelluksen kannasta.

4.3.3 Skriptien luominen

Sovelluksen päätarkoitushan oli helpottaa dynaamisten metatietojen hallinnointia, sekä edistää niiden luomisen työprosessia. Tärkeimpänä ominaisuutena voidaan siis pitää skriptien luomista, sillä tällöin päästään eroon ennen käytetyistä Excel-taulukoista. Kun käyttäjä on tehnyt tarvittavat muutokset sovelluksella, voi hän luoda tarvittavat lisäys- tai poistamisskriptit valitsemalleen attribuutille tai metaluokalle. Metaluokan lisäys- ja poistamisskripteihin sisältyy myös kaikki siihen sidotut attribuutit sekä mahdolliset valintalista-arvot. Usein sovelluksen käyttäjä saa tehtäväkseen myös muokata jo olemassa olevien metaluokkien attribuutteja tai lisätä uusia. Tällöin on tehokkaampaa luoda skriptit muutetuista ja/tai lisätyistä attribuuteista, kuin ajaa koko metaluokkaa uudestaan. Tästä johtuen sovellus mahdollistaa skriptien muodostamisen niin metaluokista, kuin niihin sidotuista attribuuteista.

Sovelluksen tietokantarakenne eroaa Twebin tietokantarakenteesta. Tästä johtuen, ei ole mahdollista hyödyntää niitä skriptejä, joilla dynaamisia metatietoja luodaan sovelluksen omaan tietokantaan. Sovellus luo asiakkaan valitseman metaluokan tai attribuutin skriptit hyödyntämällä sovelluksen päänäkymää. Kun asiakas valitsee joko metaluokka- tai attribuuttitaulusta haluamansa dynaamisen metatiedon, sovellus poimii kyseiseen metatietoon liittyvät tiedot, joita tarvitaan skriptien luomiseen. Taulunäkymäthän saavat sisältönsä listoista, jotka täytetään olioilla, jotka sisältävät tietokannasta sql-kyselyillä haetun datan. Sovellus käy listan läpi poimien tarvittavat tiedot ja lisää ne merkkijonoihin, joissa on jo valmiiksi kirjoitetut INSERT-lauseet. Kun merkkijonoihin on saatu lisättyä tarvittavat tiedot, kirjoitetaan merkkijonot sql-tiedostoon. Tämän jälkeen sovellus lataa käyttäjän päätelaitteelle kyseisen SQL-tiedoston, joka sisältää käyttäjän valitsemat skriptit.

4.3.4 Esikatselutila

Käyttäjän painaessa navigaatiopalkista kohtaa ”Esikatselutila”, aukeaa käyttäjälle uusi näkymä, josta hän voi valita metaluokan, jota esikatsella. Esikatselutilan tarkoitus on antaa käyttäjälle mahdollisuus tarkastella, miltä kyseinen dynaaminen metaluokka näyttää Twebissä. Käyttäjä valitsee metaluokan valintalistasta, joka sisältää kyseisen asiakkaan ja valitun ympäristön kaikki metaluokat. Metaluokat saadaan etsimällä metaluokkia

tietokannan taulusta, käyttäen ehtoina, että kyseessä on valitun asiakkaan ja valitun ympäristön metaluokat. Nämä metaluokat siirretään listaan ja kyseinen lista toimii tiedonlähteenä esikatselutilassa olevalle valintalistalle.

Asiakkaan valittuaan metaluokan, lataa sovellus kyseisen metaluokan kaikki attribuutit, sekä mahdolliset valintalista-arvot uuteen listaan. Tämän jälkeen sovellus käy for-looppia hyödyntäen läpi kyseisen listan. For-loopissa sovellus tarkistaa attribuutin tyypin, ja luo sitä vastaavan kentän esikatselutilaan, nimeten sen attribuutin mukaan. Käyttäjä voi halutessaan valita tarkasteltavaksi uuden metaluokan, jolloin esikatselutila päivittyy, ja näyttää uuden valitun metaluokan attribuutit kenttinä.

5 JATKOKEHITTÄMINEN

Toteutettuani sovellukselle määritetyt vaatimukset ja esiteltyäni sovellusta tilaajalle, sain jatkokehitysehdotuksia. Sovellukselle vois kehittää sisäänkirjautumisen, jonka kautta pääsisi käyttämään sovellusta. Tämä mahdollistaisi tehokkaamman lokituksen hyödyntämisen, sillä nähtäisiin, kuka on tehnyt ja mitä on tehnyt. Tällä hetkellä sovellus lokittaa kaikki muutokset, mutta tekijää ei pystytä tarkentamaan. Myös virhetilanteiden kohdalla olisi helpompaa lähestyä virheitä, kun tiedettäisiin kenen toimesta ne ovat tapahtuneet. Käyttämällä sisäänkirjautumissivua voitaisiin myös määritellä eri oikeuksia eri käyttäjille, jolloin sovelluksen helppokäyttöisyyttä ja vikasietoa saataisiin parannettua.

Esikatselutilaan ehdotettiin jatkokehitysideana mahdollisuutta tarkastella suoraan xml-tiedostosta, että miltä dynaamiset metatiedot näyttäisivät Twebissä, ennen kuin käyttäjä alkaisi tekemään muutoksia. Olisi myös kätevää, jos käyttäjä pystyisi muutosten tekemisen jälkeen vertaamaan tekemiään muutoksiaan alkuperäiseen tilaan. Tällä saataisiin varmuutta käyttäjälle, että kaikki on niin kuin pitää, paitsi tehdyt muutokset.

Nykyisen esikatselutilan voisi jakaa kahteen osaan, joista toiseen puoleen sijoittaisi tämän vertailunäkymän. Vertailunäkymästä voisi valita xml-tiedoston, jonka sovellus sitten muuttaisi suoraan Twebissä näkyviin kenttiin. Esikatselutilan logiikkaa voisi parantaa myös, sillä tällä hetkellä esikatselutila näyttää kaikki kyseisen metaluokan kentät, ottamatta kantaa riippuvuuksiin. Toki, tämä on vähän kaksipiippuinen asia, sillä jossain tilanteissa olisi hyvä nähdä kentät toimivuudeltaan niin kuin Twebissä, mutta taas joissain tilanteissa tarpeellista olisi nähdä kaikki kentät kaikkineen valintalista-arvoineen riippumatta riippuvuuksista.

Vaihtoehtona XML:n hyödyntämiseen tiedonsiirtäjänä olisi sitä tehokkaampi JSON-formaatti. Jatkokehityksen kannalta JSON:iin voisi tutustua, ja evaluoida voisiko sitä mahdollisesti käyttää vähän kankean XML:n sijaan. XML:n vaihtaminen JSON:iin vaatisi komentorivisovelluksen uudelleen tekemisen, sekä XML-käsittelystä vastaavan logiikan muuttamista, jotta tarvittavien tietojen saaminen JSON-tiedostosta onnistuu.

Tärkein askel jatkokehittämisessä on kuitenkin sovelluksen testaamisen aloittaminen. Kun saadaan joku muu, kuin itse kehittäjä käyttämään sovellusta asiakastilanteessa, saadaan paljon tärkeää dataa ja nähdään tarvittavat tarpeet muutoksille. Ennen kaikkea, käyttäjän palaute sovelluksesta on merkittävää ja se tulee ottaa huomioon. Testaamalla sovellusta voidaan sitä muokata parempaan suuntaan ja kehittää sitä entisestään tehokkaammaksi työkaluksi, jota on vaivatonta, ja jopa ilo käyttää.

6 YHTEENVETO

Opinnäytetyön aihe-ehdotus oli itselleni mieluinen, sillä halusin tutustua ohjelmistokehitykseen työelämässä. Aikaisemmat kokemukset sovelluksien kehittämisestä rajautui itselläni hyvin pitkälti koulussa toteutettuihin projekteihin ja tehtäviin, sekä pieniin omiin projekteihin, joissa jatkojalostin koulussa opittuja asioita ja tekniikoita. Ennen opinnäytetyötä, en ollut koskaan kuullutkaan Vaadin sovelluskehyksestä, puhumattakaan käyttänyt mitään sovelluskehystä ohjelmoidessani. Alkuun pääseminen ei itsessään ollut vaikeaa, sillä tilaajan puolelta löytyi Vaadin sovelluskehykseen liittyvää kirjallisuutta, joka auttoi pääsemään alkuun. Myös Vaatimen omilta sivuilta löytyy kattava määrä dokumentointia Vaadin sovelluskehykseen liittyen.

Mitä enemmän Vaadin sovelluskehystä käytin, sitä selkeämmin huomasin sovelluskehyksen tehokkuuden. Vaadin tarjoaa loistavia valmiiksi rakennettuja ominaisuuksia, sekä selkeät ja tyylikkäät komponentit, joita hyödyntämällä ohjelmointi on tehokkaampaa ja jälki näyttävämpää. Hahmotellessani käyttöliittymää sovellukselleni, en käsitellyt mitään sovelluksen muotoiluun tai tyyliin liittyvää, ja sovelluksen ulkonäkö oli yksinkertainen ja tyylitelty. Vastaavan tuloksen saaminen olisi huomattavasti työläämpää, jollei käytössä olisi Vaadin sovelluskehyksen kaltaista työkalua. Voisin ehdottomasti suositella Vaadin sovelluskehystä niin kokeneemmalle, kuin vasta Javaan tutustavalle sovelluskehittäjälle.

Vaikeinta opinnäytetyössä oli varmasti kokonaisuuksien hahmottaminen. Sekä tiedon rajaaminen, mikä riittäisi käyttäjälle ja mikä ei. Sovelluskehityksen alkuvaiheilla tuskailin myös tietokantayhteyden, sekä sen hyödyntämisen kanssa.

Motivoivaa opinnäytetyössä oli se, että kyseessä oli toiminallinen projekti, mikä pyrkii ratkomaan oikeaa pulmaa ja tehostamaan työprosessia työelämäympäristössä. Toki se, että kyseessä oli yrityksen sisäinen sovellus, antoi minulle jonkin verran vapauksia kehittäessäni sovellusta.
Tämä avasi kuitenkin silmiäni liittyen yrityksen sisäisiin toimintatapoihin ja työkaluihin, josta uskon olevan suurta hyötyä tulevaisuutta ajatellen. Sovelluksen työstäminen oli ajoittain haastavaa ja ongelmallista, mutta juuri ne pienet ongelmat ja vastoinkäymiset opettavat. Ongelmatilanteissa ohjaajani ohjasivat minua oikeaan suuntaan ja ehdottivat erilaisia lähestymistapoja ratkomaan ongelmia. Työympäristönä Triplan Oy oli loistava. Ohjaajani opetti minulle paljon ja esitti mahdollisia ratkaisuja, mutta ei ikinä tehnyt työtä puolestani, jolloin opin itse tekemällä. Vastaavasti, tarpeen ilmetessä, pystyin kysyä myös apua tai mielipidettä muilta työntekijöiltä, jotka auttoivat aina, jos vain pystyivät.

Opinnäytetyössä saavutettiin sille asetetut tavoitteet. Opinnäytetyöstä käy ilmi, että on hyvinkin mahdollista hallinnoida Twebin dynaamisia metatietoja web-pohjaisella sovelluksella. Sovelluksessa pystytään luomaan ja poistamaan dynaamisia metatietoja, sekä luomaan niistä tarvittavat skriptit. Tällöin voidaan luopua vanhoista, sekä kankeista työvaiheista ja tehostaa työprosessia.

LÄHTEET

Eclipse Foundation (n.d.). FAQ What is Eclipse? Haettu 20.7.2017 osoitteesta
https://wiki.eclipse.org/FAQ_What_is_Eclipse%3F

GWT (n.d.). Overview. Haettu 15.7.2017 osoitteesta
http://www.gwtproject.org/overview.html

Maven (n.d.). What is Maven? Haettu 19.7.2017 osoitteesta
https://maven.apache.org/what-is-maven.html

Rouse, M. Rich Internet Application (RIA). Haettu 10.7.2017 osoitteesta
http://searchmicroservices.techtarget.com/definition/Rich-Internet-Application-RIA

Vaadin Team (2017.) Book of Vaadin. Vaadin Framework 8. Volume 1. Turku: Vaadin Oy.

Vaadin Team (2017.) Book of Vaadin. Vaadin Framework 8. Volume 2. Turku: Vaadin Oy.

Vaadin (2011) IT Mill Oy on nyt Vaadin Oy. Haettu 5.7.2017 osoitteesta
https://vaadin.com/press/2011-02-23-it-mill-on-nyt-vaadin

Vaadin (2016a.) We are Vaadin. Haettu 3.7.2017 osoitteesta
https://vaadin.com/company

Vaadin (2017b.) FAQ. Vaadin. What is Vaadin? Haettu 3.7.2017 osoitteesta
https://vaadin.com/faq

Kuva 1: Vaadin (n.d.). Vaadin Application Architecture. Haettu 17.7.2017 osoitteesta
https://vaadin.com/vaadin-fw8-documentation-portlet/framework/application/img/application-architecture.png

Kuva 4: Vaadin. Vaadin Designer Views. Haettu 20.7.2017 osoitteesta:
https://vaadin.com/image/image_gallery?uuid=ec1ca697-454d-4607-8af7-5ef0d0673b50&groupId=10187&t=1305548740828

Kuva 2: W3Schools (n.d.). How AJAX Works. Haettu 17.7.2017 osoitteesta
https://www.w3schools.com/xml/ajax.gif

Wikipedia (2017 a.) Rich Internet application. Haettu 10.7.2017 osoitteesta
https://en.wikipedia.org/wiki/Rich_Internet_application

Wikipedia (2017 b.) Ohjelmistotuotanto. Haettu 22.8.2017 osoitteesta https://fi.wikipedia.org/wiki/Ohjelmistotuotanto